PATENT ABSTRACTS OF JAPAN

(11)Publication number : 11-065852
(43)Date of publication of application : 09.03.1999

(51)Int.Cl. G06F 9/45

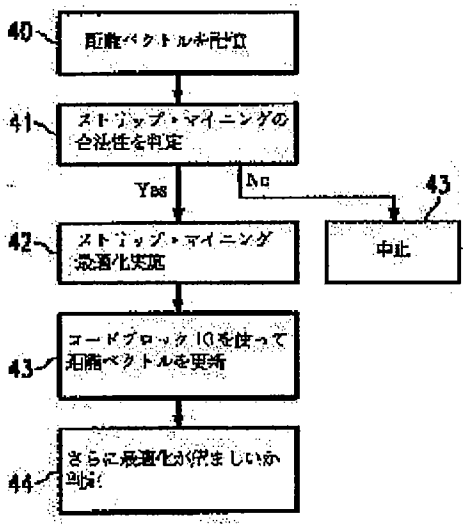
(21)Application number : 10-168205
(22)Date of filing : 16.06.1998

(71)Applicant : HEWLETT PACKARD CO <HP>
(72)Inventor : TERRY J KARAKUTTSUO

(30)Priority
Priority number : 97 879210 Priority date : 19.06.1997 Priority country : US

(54) COMPILER SYSTEM

(57)Abstract:
PROBLEM TO BE SOLVED: To make a loop compatible while evading a connected suffix system and maintaining easily and accurate compilation by providing a means which stores the distance vector of the part of a code before optimization and a means which decides the evaluation reference of optimization and updating the stored distance vector according to a given expression including the evaluation reference and stored distance vector.
SOLUTION: A compiler stores the original distance vector (40) and decides the legality of the execution of the optimization of strip mining (41). When the execution is legal, the compiler implements the strip mining optimization (42) and when not, the optimization is not performed (43). After the strip mining, the compiler uses the stored distance vector and applies the relation of a code block to update the distance vector (43). The compiler updates the original distance vector by using the relation merely shown in the code block and only determines the distance vector of a strip mining loop.



(19)日本国特許庁（ＪＰ）　(12)公開特許公報（A）　(11)特許出願公開番号

特開平11－65852

(43)公開日　平成11年(1999)３月９日

(51)Int.Cl.[6]	識別記号	ＦＩ	
Ｇ０６Ｆ　9/45		Ｇ０６Ｆ　9/44	３２２Ｇ

審査請求　未請求　請求項の数１　ＯＬ　（全 9 頁）

(21)出願番号　特願平10－168205

(22)出願日　平成10年(1998)６月16日

(31)優先権主張番号　８７９，２１０

(32)優先日　1997年６月19日

(33)優先権主張国　米国（ＵＳ）

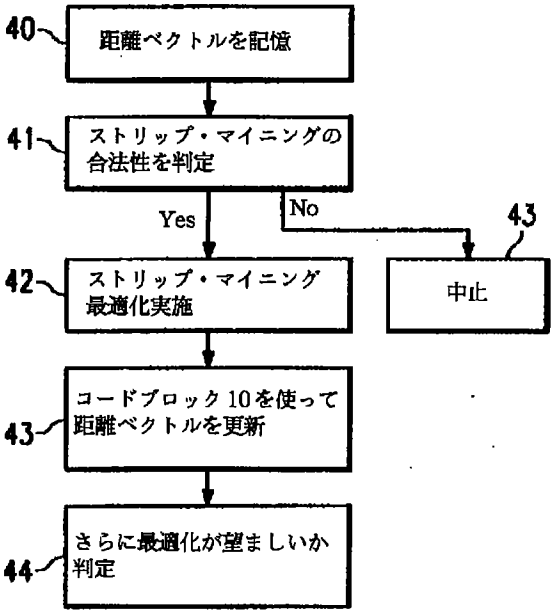
(71)出願人　398038580
ヒューレット・パッカード・カンパニー
HEWLETT－PACKARD COMPANY
アメリカ合衆国カリフォルニア州パロアルト　ハノーバー・ストリート　3000

(72)発明者　テリー・ジェイ・カラクッツオ
アメリカ合衆国75218テキサス州ダラス、ビスケイン・ブルバード　9538

(74)代理人　弁理士　岡田　次生

(54)【発明の名称】　コンパイラ・システム

(57)【要約】
【課題】コンパイラによる距離ベクトルの更新を可能とし、最適化の非静的順序づけを可能とし、従属性の正規化および再分析のオーバーヘッドの発生を防止し、連結添字式を避け、コンパイルの簡略性と正確性を維持しながらループの互換を可能とする。
【解決手段】コードのある部分の最適化の実行に先だってコードの前記部分に関する距離ベクトルを更新するコンパイラシステムであって、最適化の前にコードの前記部分の距離ベクトルを記憶する手段と、最適化の評価基準を判定する手段と、評価基準と記憶された距離ベクトルを含む所定の式にしたがって距離ベクトルを更新する手段とを備える。



1

【特許請求の範囲】
【請求項１】コードのある部分の最適化の実行に先だってコードの前記部分に関する距離ベクトルを更新するコンパイラシステムであって、
前記最適化の前にコードの前記部分の前記距離ベクトルを記憶する手段と、
前記最適化の評価基準を判定する手段と、
前記評価基準と前記記憶された距離ベクトルを含む所定の式にしたがって前記距離ベクトルを更新する手段と、
を備えるコンパイラ・システム。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】この発明は、一般的にはシステムコンパイラの最適化に関し、特にコンパイラがプログラムコード上のループストリップ化の最適化を実行した後の距離ベクトルの更新に関する。
【０００２】
【従来の技術】従属性の分析はメモリ参照順序の制約条件となる規則群であり、コンパイラによって決定される。たとえば、あるプログラムが順次実行されるときメモリ参照Bがメモリ参照Aに続く場合、あるいはAとBが同じ記憶場所を参照する場合、メモリ参照Bはメモリ参照Aに従属するものとみなされる。メモリ参照には従属性があり、メモリ参照はプログラム言語のセマンティクス（semantics：意味論）よって要求される順序で発生するよう制約される。メモリ参照の従属性には２つの割り当ての発生順序を制約する出力従属、割り当てより先に使用を発生するように制約する逆従属、割り当てを使用より先に発生するように制約するフロー従属および２つの使用の発生順序を制約する入力依存等がある。他の従属性としては、ある動作を制御フローがその動作の実行が可能なものであるかを判定するテストの後に発生するよう制約する制御従属、および入力に続いて動作が発生するように制約する動作従属がある。
【０００３】図１にはユーザーがなんらかの高級言語で書いたプログラムからなるソースファイル11を有する典型的なコンパイル環境10の全体的構造を示す。ファイル11がコンパイラ12によって処理されてオブジェクトファイル13が得られ、オブジェクトファイル13は通常ソースファイル11の高級ソース・ステートメントの翻訳結果である一連の機械命令からなる。オブジェクトファイル13はその後リンカープログラム14によって処理され、リンカープログラム14はオブジェクトファイル13を他のソースファイル（図示せず）から得られた他のオブジェクトファイル15と組み合わせて実行可能なプログラム16を生成する。実行可能なプログラム16はコンピュータ17上で直接実行することができる。したがって、このプログラムはなんらかの入力18を読み出し、処理を実行し、なんらかの出力19を生成する。従属性分析およびループの最適化は通常図１に示すコンパイラの一部として実行される。

2

【０００４】図２は図１のコンパイラ12を最適化したものの内部構造の図である。このタイプのコンパイラはソースファイル11をオブジェクトファイル13に翻訳するだけではなく、作成されたオブジェクトファイルのランタイム性能の改善をはかるものである。このコンパイラはソースファイル11から始まる。このコンパイラのフロントエンド21によってソースファイルが読み込まれ、構文エラーあるいはセマンティクスエラーがチェックされる。エラーがないものとすると、コンパイルが進行し、フロントエンド21が中間表現22を生成する。オプティマイザ23がコードの実行を高速化する変換を実行することによって、中間表現22の構造の改善そしてそれによってランタイム性能の改善をはかる。最終ステップではオブジェクトファイル13が生成され、これは通常オブジェクトファイル発生器24によってコンパイラのバックエンドで実行される。
【０００５】図３には図２に示すオプティマイザ23の内部構造を示す。このオプティマイザはコンパイルされる各プロシージャの最適化されていない低レベルの中間表現31から始めて、各プロシージャについて最適化された中間表現35を生成する。最初の段階は、実行可能な最適化を判定するための中間表現の分析32である。ここでは、コンパイルされるプロシージャのループ構造を認識し、従属性の分析を実行する。第２の段階は、コードに対してループの最適化を含むさまざまな最適化を実行し、可能なところでは距離ベクトルを更新することである。規則上、あるいは距離ベクトルが管理不能となったためにそれ以上の最適化が不可能となったとき、オプティマイザは命令のスケジューリングやレジスタの割り当てといった最適化後段階34を実行する。その結果最適化された中間表現35が得られ、これがオブジェクトファイル発生器24によって処理されてコンパイルされたオブジェクトコード13が得られる。
【０００６】プログラムの内部表現の各ノードは潜在的に多くの異なるランタイム参照を表わす。ノードNAからノードNBへの従属性のファミリーはノードNAへの潜在的なランタイム参照およびノードNAへのランタイム参照の間の従属性の集合である。ノード間でコンパイラが描く弧はかかるファミリーの表現である。たとえば、コードブロック（CODE BLOCK）1に示すループについては、コンパイラはx(i+1) = からx(i)への弧を描き、それにラベルを付してコードブロック2に示す従属性のファミリーを表現する。
【０００７】
【表１】

CODE BLOCK 1

```
do i = 1,n
```

```
        x(i+1)=x(i)*y(i)
  enddo
CODE BLOCK 2
  x(2)= -->x(2)
  x(3)= -->x(3)
  x(4)= -->x(4)
  ...
CODE BLOCK 3
  do i = 1,n
       do j = 1,n
            do k = 1,n
                 x(i+1,j+2,k+3)= 0
            enddo
       enddo
  endodo
```

【０００８】メモリ参照はある数の周囲のループ内で発生する。反復ベクトル識別子がそれらのループのどの反復である特定のノードによる特定のメモリ参照を実行するかを同定する。そのボードにnの周囲ループがある場合、ある参照の反復ベクトルはnタプル（tuple）である。このNタプルの各要素は対応する周囲ループの反復数であり、最も外側のループは最初の要素に対応し、以下同様である。たとえば、コードブロック3は、３深の入れ子ループすなわちDo i、Do j、Do kを有するプログラムフラグメントを示し、各ループは１ずつステップし、各ループの上限はnである。ループ本体はxと呼ばれる＊る配列の３次元配列要素への１つの参照からなる。１次元の添字はi+1、２次元はj+2、３次元はk+3である。割り当てx(3,7,4)に対する反復ベクトルは、０原点反復ベクトルの場合（1,4,0）である。１原点反復ベクトルの場合、ベクトルは（2,5,1）である。０原点を用いるとすべての可能なプログラム言語に対応可能である。フォートランは、１原点を用いており、適当にスキューされる。

```
CODE BLOCK 4
  do i = 1,n
       do j = 1,n
            y(i+3,j-1)=y(i,j)
       enddo
  endodo
```

【０００９】

【表２】

【００１０】距離ベクトルは２つの反復ベクトルの間の差である。たとえば、表２のコードブロック4は２深の入れ子型ループすなわちDo i、Do jを有するプログラムフラグメントを示し、各ループは１ずつステップし、各ループの上限はnである。ループ本体は２次元配列yへの割り当てへの１つの参照からなり、y(i+3j-1) = y(i,j)である。y(4,4)=およびy(4,4)に対する反復ベクトルはそれぞれ（4,4）および（3,3）である。距離ベクトルは（3,3）マイナス（0,4）であり、これは（3,-1）である。ある符号規定については、宛先に対する反復ベクトルが減算の最初に来る。符号規定は分析全体を通して一貫して適用されるかぎりは重要ではない。距離ベクトルは、いつも、非単位ストライドを有するループにつては添字の減算によって計算しうるとは限らないことに注意されたい。

【００１１】方向ベクトルは距離ベクトルの符号である。つまり、方向ベクトルは距離ベクトルの各要素を+1,0および/または-1で置き換えることによって実現される。たとえば、（3,-1）の距離ベクトルは（+1,-1）の対応する方向ベクトルを有する。方向ベクトルは通常数字ではなく記号を用いて書かれる。+1は“<”で、０は“=”で、-1は“>”で書かれる。したがって、（3,-1）の距離ベクトルは（<,>）で表わされる。“>”記号は反復回数の増大を、“<”記号は反復回数の減少を示す。

【００１２】従属性のファミリーは均一であることが多く、これはそのファミリーの従属性は同じ距離ベクトルと方向ベクトルを有することを意味する。CODE BLOCK 4のループについていえば、y(i+3j-1)=からy(i,j)への従属性ファミリーは均一である。したがって、（3,-1）はファミリ距離ベクトル、（<,>）はファミリー方向ベクトルとみなすことができる。

【００１３】スーパーコンピュータのコンパイラには機械のアーキテクチャをより有効に利用するためにユーザーによって入力されたプログラムコードを実際に変換する最適化技術が用いられる。たとえば、並列処理機械の

場合、ステップ２の完了を待ってステップ３に移行するように動作を順次実行する代わりに、機械はプログラムを書き直してステップ２とステップ３を同時に処理することを可能とする。コンパイラは従属性分析を実行してこのタイプの変換が合法であるかどうかを判定する。この変換が合法である場合、ユーザーは速度をたとえば５倍にも上げることができ、しかも従属性分析からその結果が正しいものであることが保証される。変換が不法なものである場合、コンパイラはそのコードをそのまま実行し、変換を行なわない。

【００１４】コンパイラが最適化しうる領域の１つとしては、反復構造すなわちループがある。たとえば、あるプログラムの一部がマトリクスの異なる行と列を乗算するマトリクス乗算を実行するように書かれる。このコンパイラはさまざまな変換を用いることによって、コードのその部分をより高速にランしながら正しい結果が得られるように書き直しうることを判定することができる。

【００１５】変換を実行する前に、コンパイラはプログラム中のすべてのループネストおよび各ネストにおける参照対について従属性分析を実行して反復スペースと周囲ループ内での参照の持続時間中での参照のメモリ制約条件を判定する。この情報は通常コンパイラに距離（すなわち従属性）ベクトルおよび方向ベクトルとして記録される。距離ベクトルの長さは参照のおよぶ共通するループの数であり、ベクトル内の各要素はメモリ従属距離である。方向ベクトルは２つの参照の反復ベクトルから距離がただちに判定できない場合に使用することができる。

【００１６】このコンパイラが通常実行するループ変換の１つは距離ベクトルを用い、ループ・ストリップ・マイニング（loop strip-mining）あるいはループ・ブロッキング（loop blocking）と呼ばれる。このループ変換によれば、ループを部分ごとに実行することによって＊ループのオーバーヘッドが低減される。これによって、ループを処理するさいにキャッシュメモリをより多く再使用することができる。通常、機械はレジスタ以外にプロセッサに非常に近く配置されたキャッシュメモリシステム（たとえばL2キャッシュ）を有し、遠隔のRAMメモリよりも待ち時間が短い。しかし、キャッシュの再ローディングを行なう場合時間および性能上のペナルティが生じる。このペナルティはコスト上昇につながり、キャッシュロード数を低減する方が有益である。

【００１７】ストリップ・マイニングは、ループがキャッシュ内に収まるように変換することによってキャッシュロードの数を低減する。ループは内側のループが各キャッシュ行にアクセスし、外側のループがキャッシュ全体にアクセスするように追加のループを加えることによって変換される。このタイプの変換はスーパーコンピュータにおけるベクトル処理から並列処理にいたるまで広く用いられ、キャッシュメモリの待ち時間を短縮している。この変換はプログラムコードになかった追加ループを導入し、変換されたループの配列要素の添字をもとのループと新たなループの両方の関数になるように変更する。

【００１８】コードブロック5および6にストリップ・マイニングすなわちブロッキングの例を示し、これらはそれぞれブロッキングすなわちストリップ・マイニングの後２ループのネストとなる１ループのネストを示す。コンパイラは、各ループネストについてプログラム全体の従属性をいったん計算する。これは、実際にはペア型のアルゴリズムを用いる非常にコストのかかる処理であり、これはXの参照については約X^2の可能性があることを意味する。

【００１９】

【表３】

```
CODE BLOCK 5
   do i = 1,10
      a(i+3) = ...
      ... = a(i)
CODE BLOCK 6
   do j = 1,10,4
      do i = j,min(j+4-1,10)
         a(i+3) = ...
         ... = a(1)
```

【００２０】コードブロック5は、Do i = 1～10、a(i+3) =であり、内部でa(i)を用いる簡単なループを示す。このコンパイラはコンピュータシステムの特性、特にループのサイズと比較したキャッシュのサイズに基づいてこのループを４つの部分に分けて実行することがより効率的であると判断する。したがって、外側のループが追加され、内側のループの項がコードブロック6に示すように変更され、もとのコードは、Do j = 1～10×４およびDo i = j～（j+4-1,10）のうちの最小値、に変換され、a(i+3)はa(i)である。したがって、jの値は１、５および９となる。iの値はjとともに変化し、j=1であるときiの値は1-4、j=5であるときiの値は5-8、j=9であるときiの値は9-10となる。こうしてステップサイズは次の態様で１から４まで変えられる。すなわち、ステップサイズ長である最大４か、あるいはループ反復数がストリップサイズで割り切れない場合にはその剰余まで内側

のループが外側のループのストリップを実行する。外側のループをセクションループと呼び、内側のループを要素ループと呼ぶ。

【００２１】

【発明が解決しようとする課題】従来のコンパイラにおいては、この変換を実行した後にはコンパイラはこのループではそれ以上機能しえず、その結果ブロックされた後にはループを最適化することはできない。これは添字が理解しずらく複雑なものになっているためである。簡単な添字であったものが追加のループによって複雑なも＊のとなっている。さらに、追加ループはもとのコードには存在しておらず、これはもとの距離ベクトルのサイズを再度求めるすなわち計算し直すことが必要であることを意味する。これは、実行すべき従属性分析については、各ループを正規化しなければならず、すなわち各ループが単位に基づき、誘導変数が１で始まり１ずつステップするためにさらに複雑化する。

【００２２】

【表４】

```
CODE BLOCK 7
     do j = 1,3
        do i = (j-1)*4+1, min((j-1)*4+4, 10)
           a(i+3) = ...
           ... = a(i)
CODE BLOCK 8
     do j = 1,3
        do i = 1, min(4, 4*j mod 10)
           a(4j+1-1) = ...
           ... = a(4j+i-4)
```

【００２３】たとえば、コードブロック6に示すループネストはコードブロック7および8に示すように正規化される。コードブロック7に示すように外側のループが最初に正規化され、その後コードブロック8に示すように内側のループが正規化される。コードブロック8に示すループネストは従来のコンパイラの従属性分析の入力となる。添字式は連結添字であり、これは各添字が内側と外側のループの誘導変数の両方の関数であることを意味する。これは、距離ベクトルを判定するための簡単な減算を、２つ以上の未知数の解を求めなければならない複雑な計算にする。また、一定した反復スペースすなわち従属距離であったもの（これはA(i+3)とA(i)の差すなわち３であった）が、変数になり、誘導変数iおよびjの値に依存するようになる。

【００２４】従来技術における主たる問題点は、コンパイラがブロッキングまたはストリップ・マイニングを最終のループ変換として実行することである。これは、これ以後はループ境界がmin関数およびmod関数を含むため、この点以後はいかなる最適化の合法性もテストすることが困難なためである。また、１つのループ誘導経変※数の関数である添字はこの段階では複数のループの関数である。従来のコンパイラは、この変換の後に従属性を計算できるとしても、その計算式に多数の新たな変数があるため保守的過ぎる結果を出し、その再計算のために多大なコストが生じる。

【００２５】さらに、コンパイラが変換のたびに従属性の再計算を行なわねばならないとしたら、その後変換が行なわれるたびに、ループ境界添字が各変換のセマンティクスのために複雑化するため、従属性の情報が失われる可能性が非常に高い。従来のコンパイラは従属性を前もって一度計算し、その後最適化変換の静的な順序づけを実行する。このコンパイラは２つ以上のループ誘導変数の関数である１つの添字があると最適化を停止する。このとき、コンパイラはレジスタを割り当て、機械コードを発する。距離ベクトルおよび方向ベクトルは、コードブロック8の連結添字式から計算するとしたら、次のようになるであろう。

【００２６】

【表５】

CODE BLOCK 9

	距離	方向
(1,1)-->(1,4)	(0,3)	(=,<)
(1,2)-->(2,1)	(1,-1)	(<,>)
(1,3)-->(2,2)	(1,-1)	(<,>)
(1,4)-->(2,3)	(1,-1)	(<,>)
(2,1)-->(2,4)	(0,3)	(=,<)

【００２７】従来のコンパイラは、正規化のオーバーヘッドコストと連結添字添字式の検討の困難さのためにコードブロック9に示す結果を計算しないことを指摘しておく。コードブロック9は後述する本発明との比較のために示すものである。

【００２８】したがって、当該技術分野において、コン

パイラによる距離ベクトルの更新を可能とし、最適化の非静的順序づけを可能とし、従属性の正規化および再分析のオーバーヘッドの発生を防止し、連結添字式を避け、コンパイルの簡略性と正確性を維持しながらループの互換を可能とすることが必要とされている。

【００２９】

【課題を解決するための手段】以上およびその他の目的、特徴および技術的利点は２つの関係を用いてコンパイラがループ・ストリップ・マイニングを実行した後に距離ベクトルを更新するシステムおよび方法によって達成される。この発明のコンパイラ・システムは、次の構成をとる。すなわち、コードのある部分の最適化の実行に先だってコードの前記部分に関する距離ベクトルを更新するコンパイラシステムであって、 最適化の前にコードの前記部分の距離ベクトルを記憶する手段と、最適化の評価基準を判定する手段と、評価基準と記憶された距離ベクトルを含む所定の式にしたがって距離ベクトルを更新する手段と、を備えるコンパイラ・システム。

【００３０】本発明はブロッキングされていないループについてはもとの距離ベクトルを入力とし、その距離をストリップサイズで割った結果が自然数となるかどうかによって１つまたは２つの距離ベクトルを出力する。したがって、本発明は後続の最適化の発生を可能とし、変換されたループの正規化および添字の再計算を不要とする。また、本発明はループの従属性の再分析（これは上述したN^2の計算であり変換自体によってその入力が複雑化している）を不要とする。

【００３１】さらに、本発明の出力は最適化に関する任意の合法性試験によって検査することのできる標準的な距離ベクトルフォーマットになっている。また、入力もまた距離ベクトルであり出力と同じ形式であり、これによってブロッキング変換は後続の最適化に対して透明となる。これは、出力が後続の変換の後続の合法性試験への入力となるためである。本発明の正確性と均一性によって、最適化の非静的な順序づけが可能となり、コンパイラはループをブロッキングすることができ、またそれを交換、再交換および再ブロッキングすることが可能である。また、本発明は、添字が２つ以上の誘導変数をもち、２以上の添字位置に現われる連結添字を防止する。本発明は、変換された添字式から距離ベクトルを再計算していくのではなく、もとの距離ベクトルのみを用いる。

【００３２】以上は次の本発明の詳細な説明の理解をたすけるために、本発明の特徴および技術的利点の概要を説明したものである。本発明の他の特徴および利点については以下に説明され、それらは本発明の特許請求の対象となる。当業者には、ここに開示する概念および具体的実施形態は本発明の目的を達成するための変更および他の構造の設計の基礎として容易に利用しうるものであることが理解されよう。また、当業者にはかかる均等な構造は特許請求の範囲に定める本発明の精神および範囲から逸脱するものではないことも理解されよう。

【００３３】

【発明の実施の形態】本発明は、２つの関係を用い、ブロッキングされていないループのもとの距離ベクトルをブロッキングされたすなわちストリップ・マイニングされたループの距離ベクトルである１つあるいは２つの距離ベクトルに更新する。得られる距離ベクトルが１つになるか２つになるかは、その距離がストリップサイズによって均等に割り切れて自然数となるかどうかによって決まる。本発明はループのブロッキングすなわちストリップ・マイニングのための従属性の再計算を必要としない。コンパイラは単にコードブロック10に示す関係を用いてもとの距離ベクトルを更新してストリップ・マイニングされたループの距離ベクトルを決定するだけでよい。

【００３４】

【表６】

```
CODE BLOCK 10
   (floor(d/ss), d mod ss)(ceiling(d/ss), d-(ceiling(d/ss)*ss))
CODE BLOCK 11
   (0,d)(1,d-ss)
CODE BLOCK 12
   (d/ss,0)
```

【００３５】図４に示すように、コンパイラはもとの距離ベクトルを記憶し（40）、ストリップ・マイニング最適化の実行の合法性を判定する（41）。合法である場合、コンパイラはストリップ・マイニング最適化を実行し（42）、合法でなければ最適化を実行しない（43）。ストリップ・マイニングの後、コンパイラは記憶された距離ベクトルを用い、コードブロック10の関係を適用して距離ベクトルを更新する（43）。距離ベクトルの更新は最適化が合法である場合には最適化の前に行なうことができる。コードブロック10において、第１の関係には２つの要素があり、第１の要素は距離ベクトルをストリップサイズで割った下限（floor）を決定するものであることに注意されたい。下限関数は入力の値以下の最大の整数を返す。たとえば、5/2の下限（ 5/2 と表記する場合もある）は２である。ストリップサイズすなわちブロック・マイニング係数はストリップ・マイニングによって作成される外側のループの幅である。第１の関係の第２の要素は距離ベクトルをステップサイズで割ったモジューロである。モジューロ関数（すなわちmod）は第１の入力を第２の入力で割った余りを返す。たとえ

ば、10 mod 3 は１である。また、第２の関係は２つの要素を有し、その１つが距離ベクトルをストリップサイズで割った上限を決定する。上限関数は入力の値以上で最小の整数を返す。たとえば5/2の上限（ 5/2 と表記する場合もある）は３である。

【００３６】以下はコードブロック 10をコードブロック 5および6に適用した例である。コードブロック 5の添字式a(i+3)＝a(i)はa(4)=の割り当てに対する反復ベクトル（０）とa(4)の反復ベクトル（３）を有する。したがって、+3-0は距離ベクトル（３）に等しい。次に、この距離ベクトルがコードブロック 6で用いられる４のストリップサイズとともにコードブロック 10の２つの関係に適用される。したがって、3/4の下限は０、3/4の上限は１、３を４で割ったモジューロは３となる。これによって、コードブロック 6のストリップ・マイニングされたコードの更新された距離ベクトル（0,3）および（1,-1）が得られる。この結果はコードブロック 9の結果と一致する。

【００３７】この例はまたコードブロック 10の関係の２つの特殊なケースの１つを示す。距離ベクトルがストリップサイズ未満であるとき、d/ssの係数は常に１未満である。したがって、d/ssの下限は常に０であり、d/ssの上限は常に１であり、dをssで割ったモジューロは常にdである。したがって、コードブロック 10の関係は常にコードブロック 11に示す関係に還元できる。この結果は上で得られた結果と一致する。

【００３８】第２の特殊なケースは距離ベクトルがストリップサイズの倍数であるとき、たとえば距離ベクトルが４でストリップサイズが２であるときに発生する。このような場合、d/ssの下限がd/ssの上限（これはd/ssと等しい）と等しくなる。dはssで割り切れるため、モジューロは０である。コードブロック 10の第１および第２の関係はいずれもコードブロック 12のコードに還元される。第１および第２の関係から同じ結果が生じるため、ストリップマイニングを行なったときもとの距離ベクトルから１つの従属性しか生成されない。この特殊なケースについて別の見方をすると、d/ssが自然数すなわち計数（１、２、３、…）となる場合、コードブロック 12のコードはコンパイラによる距離ベクトルの更新に用いられる。

【００３９】距離ベクトルについて、（<）の方向ループ（正の距離からの）は従属距離がストリップサイズ未満である場合（=,<）および（<,>）となる。この結果はコードブロック 11のコードと一致する。（<）の方向ループは従属距離がストリップサイズより大きい（しかし、コードブロック 12のようにストリップサイズの倍数ではない）場合（<,<）および（<,>）となる。したがって、（<）の方向ループに対する一般的なケースは（<=,<）および（<,>）である。

【００４０】更新関係への入力およびこれからの出力はいずれも距離ベクトルである。上に示したように、ある距離dについて、ループがストリップ・マイニングされると、従属距離関係によって１つあるいは２つの従属性が生成され、したがって２つの記憶場所の間の関係について２つ以上の距離ベクトルをもつことがある。２つの従属性が生成されると、それらを１つの従属性ベクトルにまとめることができる。しかし、あいまいさを避けるには、それらを区別しておく方がよい。また、ストリップ・マイニングされていないループのもとの距離ベクトルに１つの要素しかなかった場合、更新された距離ベクトルには２つの要素があることに注意を要する。これは、コンパイラがコードへのブロッキングのために新たなループを導入したためである。ユーザーがこのループを書かなかった場合にも、ストリップ・マイニングの後、コードのこの部分は２ネストのループとなる。したがって、ブロッキングまたはストリップ・マイニングが実行されるたびに追加のループがプログラムコードに作成される。

【００４１】必要であれば従属性距離ベクトルを計算し直すことなく後続の最適化を容易に実行することができる（44）。コンパイラは更新された距離ベクトルを用いて後続の最適化の合法性をチェックする。これによって最適化ルーチンの非静的な順序づけが可能である。したがって、コードブロック 10に示す本発明の関係とコードブロック 11および12に示す２つの特殊なケースとによって、コードブロック 7、8および9に示す更新された距離ベクトルの計算に係るオーバーヘッドコストを避けることができる。また、本発明の関係によれば簡略性、効率および正確性を維持しながらコードブロック 8に示す連結添字の処理を避けることができる。もとの距離ベクトルはストリップ化最適化の前に記憶され、本発明の関係によって更新される。距離および方向ベクトルは通常レジスタに記憶される。

【００４２】本発明とその利点を詳細に説明したが、特許請求の範囲に規定する本発明の精神および範囲から逸脱することなくさまざまな変更、代替および改変が可能であることが理解されよう。この発明は、例として次の実施態様を含む。

（１）コードのある部分の最適化（33）の実行に先だってコードの前記部分に関する距離ベクトルを更新するコンパイラシステム（12）であって、前記最適化の前にコードの前記部分の前記距離ベクトルを記憶する手段（43）、前記最適化の評価基準を判定する手段、および前記評価基準と前記記憶された距離ベクトルを含む所定の式にしたがって前記距離ベクトルを更新する手段（43）からなることを特徴とするコンパイラ・システム。

（２）前記評価基準はストリップサイズであり、前記最適化はループのストリップ・マイニング（42）であることを特徴とする上記１記載のコンパイラシステム。

（３）前記所定の式は第１および第２の関係からなるこ

とを特徴とする上記２記載のコンパイラシステム。

（４）前記第１の関係は第１および第２の要素を含み、前記第１の要素は下限関数からなり、前記第２の要素はモジューロ関数からなることを特徴とする上記３記載のコンパイラシステム。

（５）前記第１の要素は前記記憶された距離ベクトルを前記ストリップサイズで割った下限関数であり、前記第２の要素は前記記憶された距離ベクトルを前記ストリップサイズで割ったモジューロ関数であることを特徴とする上記４記載のコンパイラシステム。

（６）前記第２の関係は第１および第２の要素を含み、両方の要素が上限関数からなることを特徴とする上記３記載のコンパイラシステム。

（７）前記第１の要素は前記記憶された距離ベクトルを前記ストリップサイズで割った上限関数であることを特徴とする上記６記載のコンパイラシステム。

（８）前記第２の要素は前記記憶された距離ベクトルから、前記ストリップサイズと前記記憶された距離ベクトルを前記ストリップサイズで割った前記上限関数との積を引いた差であることを特徴とする上記６記載のコンパイラシステム。

（９）前記記憶された距離ベクトルは前記ストリップサイズの倍数であり、前記所定の式は第１および第２の要素からなり、前記第１の要素は前記記憶された距離ベクトルを前記ストリップサイズで割ったものであり、前記第２の要素は０であることを特徴とする上記２記載のコンパイラシステム。

（10）前記記憶された距離ベクトルは前記ストリップサイズ未満であり、前記所定の式は第１および第２の関係からなり、前記第１の関係は第１および第２の要素からなり、前記第１の関係の前記第１の要素は０であり、前記第１の関係の前記第２の要素は前記記憶された距離ベクトルであり、前記第２の関係は第１および第２の要素からなり、前記第２の関係の前記第１の要素は１であり、前記第２の関係の前記第２の要素は前記記憶された距離ベクトルから前記ストリップサイズを引いたものであることを特徴とする上記２記載のコンパイラシステム。

【００４３】

【発明の効果】この発明によると、連結添字式を避け、コンパイルの簡略性と正確性を維持しながらループの互換を可能とすることができる。

【図面の簡単な説明】

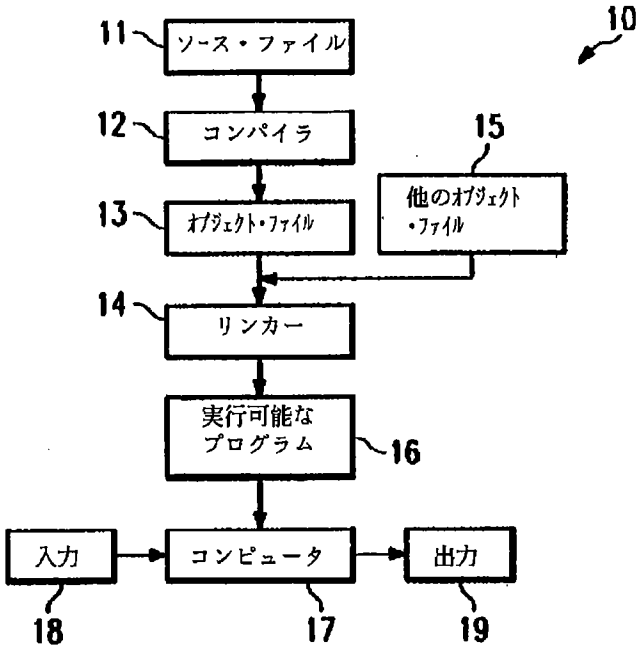
【図１】コンパイル環境の全体的構成を示すブロック図。

【図２】図１のコンパイラの内部構造を示すブロック図。

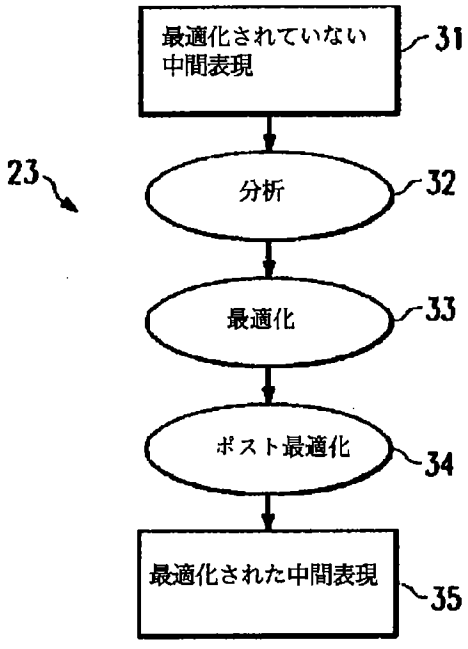
【図３】図２のオプティマイザの内部構造を示すブロック図。

【図４】本発明の関係を用いたストリップ・マイニングの概略を示すブロック図。

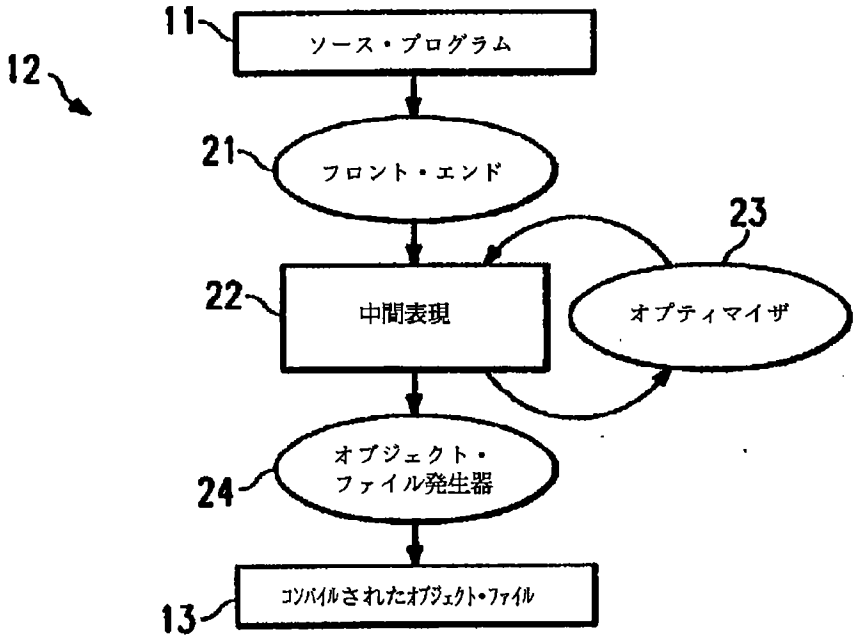
【図１】



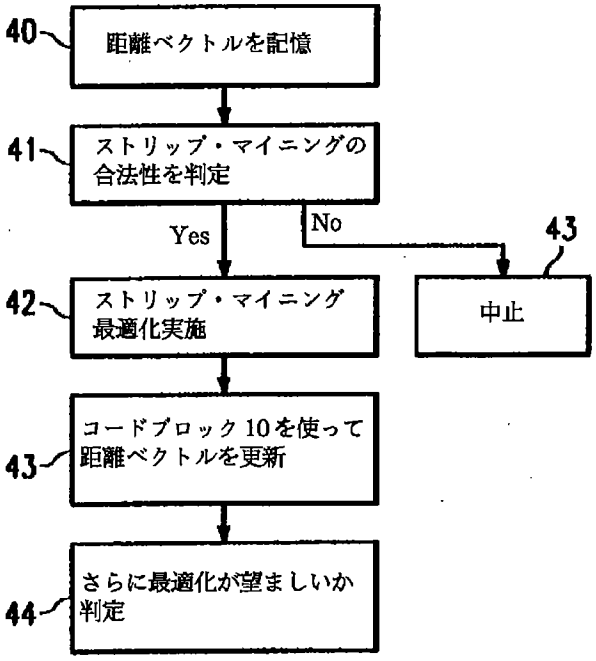
【図３】



【図２】


11
ソース・プログラム
12
21
フロント・エンド
23
22
中間表現
オプティマイザ
24
オブジェクト・
ファイル発生器
13
コンパイルされたオブジェクト・ファイル

【図４】


40
距離ベクトルを記憶
41
ストリップ・マイニングの
合法性を判定
Yes
No
43
42
ストリップ・マイニング
最適化実施
中止
コードブロック10を使って
43
距離ベクトルを更新
さらに最適化が望ましいか
44
判定